Module-2

Python String:

Python string is the collection of the characters surrounded by single quotes, double quotes, or triple quotes. The computer does not understand the characters; internally, it stores manipulated character as the combination of the 0's and 1's.
In Python, strings can be created by enclosing the character or the sequence of characters in the quotes. Python allows us to use single quotes, double quotes, or triple quotes to create the string.

Consider the following example in Python to create a string.

Syntax:

```
str = "Hi Python !"
```

Here, if we check the type of the variable **str** using a Python script
print(type(str)), then it will **print** a string (str).

Creating String in Python:

We can create a string by enclosing the characters in single-quotes or double- quotes. Python also provides triple-quotes to represent the string, but it is generally used for multiline string.

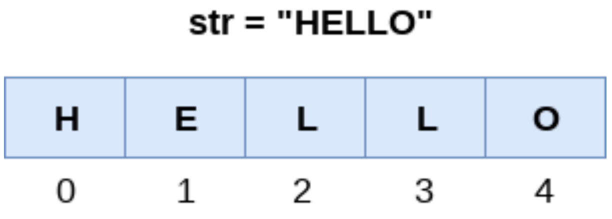
```
#Using single quotes
str1 = 'Hello Python'
print(str1)

#Using double quotes
str2 = "Hello Python"
print(str2)

#Using triple quotes
str3 = '''Triple quotes are generally used for
   represent the multiline or
   docstring'''
print(str3)
```

Output:

```
Hello Python
Hello Python
Triple quotes are generally used for
   represent the multiline or
   docstring
```

Strings indexing:

Like other languages, the indexing of the Python strings starts from 0. For example, The string "HELLO" is indexed as given in the below figure.



str[0] = 'H'

str[1] = 'E'

str[2] = 'L'

str[3] = 'L'

str[4] = 'O'

Consider the following example:

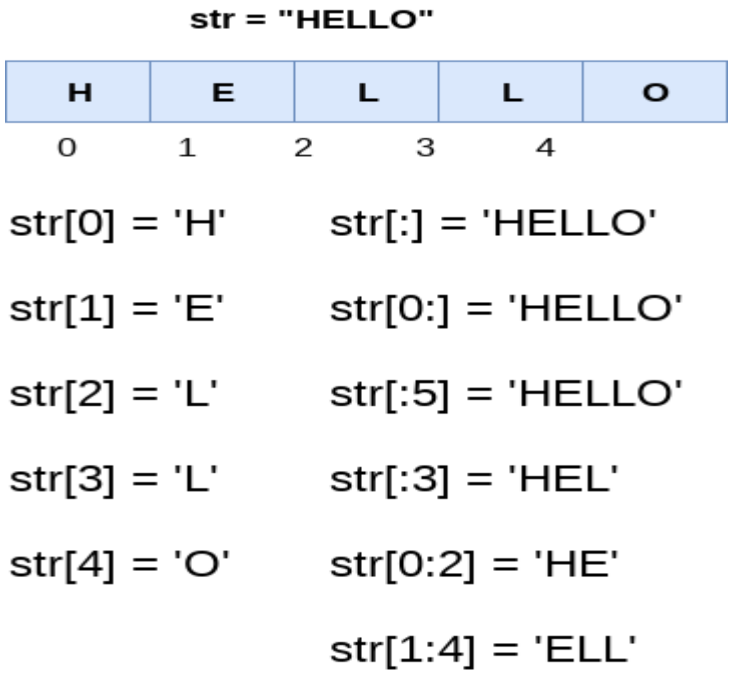
```
str = "HELLO"
print(str[0])
print(str[1])
print(str[2])
print(str[3])
print(str[4])
# It returns the IndexError because 6th index doesn't exist
print(str[6])
```

Output:

```
H
E
L
L
O
IndexError: string index out of range
```

Slicing:

You can return a range of characters by using the slice syntax. Specify the start index and the end index, separated by a colon, to return a part of the string. As shown in Python, the slice operator **[]** is used to access the individual characters of the string. However, we can use the **: (colon)** operator in Python to access the substring from the given string. Consider the following example.



```
str[0] = 'H'        str[:] = 'HELLO'
str[1] = 'E'        str[0:] = 'HELLO'
str[2] = 'L'        str[:5] = 'HELLO'
str[3] = 'L'        str[:3] = 'HEL'
str[4] = 'O'        str[0:2] = 'HE'
                    str[1:4] = 'ELL'
```

Here, we must notice that the upper range given in the slice operator is always exclusive i.e., if str = 'HELLO' is given, then str[1:3] will always include str[1] = 'E', str[2] = 'L' and nothing else.

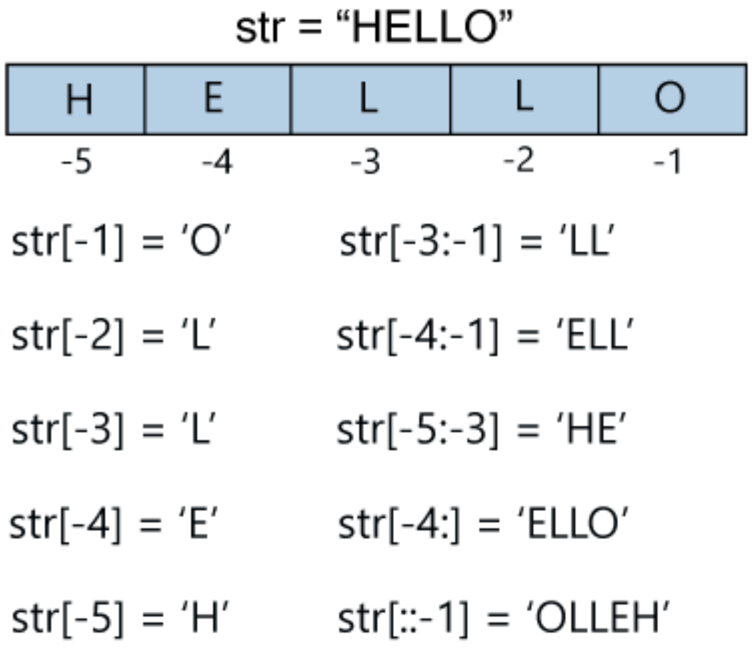
Consider the following example:

```
# Given String
str = "JAVATPOINT"
# Start 0th index to end
print(str[0:])
# Starts 1th index to 4th index
print(str[1:5])
# Starts 2nd index to 3rd index
print(str[2:4])
# Starts 0th to 2nd index
print(str[:3])
#Starts 4th to 6th index
print(str[4:7])
```

Output:

```
JAVATPOINT
AVAT
VA
JAV
TPO
```

We can do the negative slicing in the string; it starts from the rightmost character, which is indicated as -1. The second rightmost index indicates -2, and so on. Consider the following image.



Consider the following example

```
str = 'JAVATPOINT'
print(str[-1])
print(str[-3])
print(str[-2:])
print(str[-4:-1])
print(str[-7:-2])
# Reversing the given string
print(str[::-1])
print(str[-12])
```

Output:

```
T
I
NT
OIN
ATPOI
TNIOPTAVAJ
IndexError: string index out of range
```

Reassigning Strings:

Updating the content of the strings is as easy as assigning it to a new string. The string object doesn't support item assignment i.e., A string can only be replaced with new string since its content cannot be partially replaced. Strings are immutable in Python.

Consider the following example.

Example 1:

```
str = "HELLO"
str[0] = "h"
print(str)
```

Output:

```
Traceback (most recent call last):
  File "12.py", line 2, in <module>
    str[0] = "h";
TypeError: 'str' object does not support item assignment
```

However, in example 1, the string **str** can be assigned completely to a new content as specified in the following example.

Example 2:

```
str = "HELLO"
print(str)
str = "hello"
print(str)
```

Output:

```
HELLO
hello
```

Deleting the String:
As we know that strings are immutable. We cannot delete or remove the characters from the string. But we can delete the entire string using the **del** keyword.

```
str = "JAVATPOINT"
del str[1]
```

Output:

```
TypeError: 'str' object doesn't support item deletion
```

Now we are deleting entire string.

```
str1 = "JAVATPOINT"
del str1
print(str1)
```

Output:

```
NameError: name 'str1' is not defined
```

String Operators:

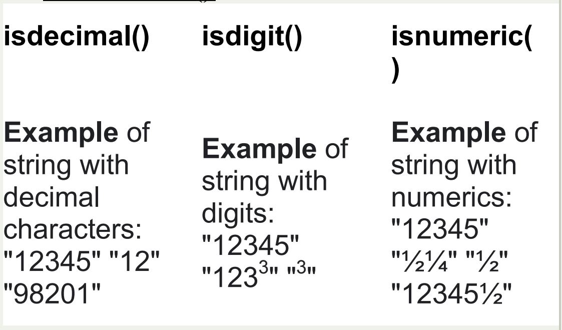
Operator	Description
+	It is known as concatenation operator used to join the strings given either side of the operator.
*	It is known as repetition operator. It concatenates the multiple copies of the same string.
[]	It is known as slice operator. It is used to access the sub-strings of a particular string.
[:]	It is known as range slice operator. It is used to access the characters from the specified range.
in	It is known as membership operator. It returns if a particular sub-string is present in the specified string.
not in	It is also a membership operator and does the exact reverse of in. It returns true if a particular substring is not present in the specified string.

Example:
Consider the following example to understand the real use of Python operators.

```
str = "Hello"
str1 = " world"
print(str*3) # prints HelloHelloHello
print(str+str1)# prints Hello world
print(str[4]) # prints o
print(str[2:4]); # prints ll
print('w' in str) # prints false as w is not present in str
print('wo' not in str1) # prints false as wo is present in str1.
```

Python String functions:
Python provides various in-built functions that are used for string handling.

Method	Description
capitalize()	It capitalizes the first character of the String. This function is deprecated in python3
count(string,begin,end)	It counts the number of occurrences of a substring in a String between begin and end index.

find(substring,beginIndex, endIndex)	It returns the index value of the string where substring is found between begin index and end index.
isalnum()	It returns true if the characters in the string are alphanumeric i.e., alphabets or numbers and there is at least 1 character. Otherwise, it returns false.
isalpha()	It returns true if all the characters are alphabets and there is at least one character, otherwise False.
isdecimal()	It returns true if all the characters of the string are decimals.
isdigit()	It returns true if all the characters are digits and there is at least one character, otherwise False.
isnumeric()  **isdecimal()** — **Example** of string with decimal characters: "12345" "12" "98201" **isdigit()** — **Example** of string with digits: "12345" "123^3" "^{3}" **isnumeric( )** — **Example** of string with numerics: "12345" "½¼" "½" "12345½"	It returns true if the string contains only numeric characters.
islower()	It returns true if the characters of a string are in lower case, otherwise false.
isupper()	It returns true if characters of a string are in Upper case, otherwise False.
lower()	It converts all the characters of a string to Lower case.
len(string)	It returns the length of a string.
swapcase()	It inverts case of all characters in a string.
title()	It is used to convert the string into the title-case i.e., The string **meEruT** will be converted to Meerut.
upper()	It converts all the characters of a string to Upper Case.

Python List:

A list in Python is used to store the sequence of various types of data. Python lists are **mutable** type, its mean **we can modify its element** after it created. However, Python consists of **six data-types** that are capable to store the sequences, but the most common and reliable type is the **list**.
A list can be defined as a collection of values or items of different types. The items in the list are separated with the comma "**,**" and enclosed with the square brackets [].

A list can be define as below

```
1. L1 = ["John", 102, "USA"]
2. L2 = [1, 2, 3, 4, 5, 6]
```

If we try to print the type of L1 and L2 using type() function then it will come out to be a list.

```
1. print(type(L1))
2. print(type(L2))
```

Output:

```
<class 'list'>
<class 'list'>
```

Characteristics of Lists:

The list has the following characteristics:

- The lists are ordered.
- The element of the list can access by index.
- The lists are the mutable type.
- The list elements are mutable types.
- A list can store the number of various elements.
- Since lists are indexed, lists can have items with the same value.

Let's check the first statement that lists are the ordered.

```
a = [1,2,"Peter",4.50,"Ricky",5,6]
b = [1,2,5,"Peter",4.50,"Ricky",6]
a ==b
```

Output:

```
False
```

Both lists have consisted of the same elements, but the second list changed the index position of the 5th element that violates the order of lists. When compare both lists it returns the false.
Lists maintain the order of the element for the lifetime. That's why it is the ordered collection of objects.

```
a = [1, 2,"Peter", 4.50,"Ricky",5, 6]
b = [1, 2,"Peter", 4.50,"Ricky",5, 6]
a == b
```

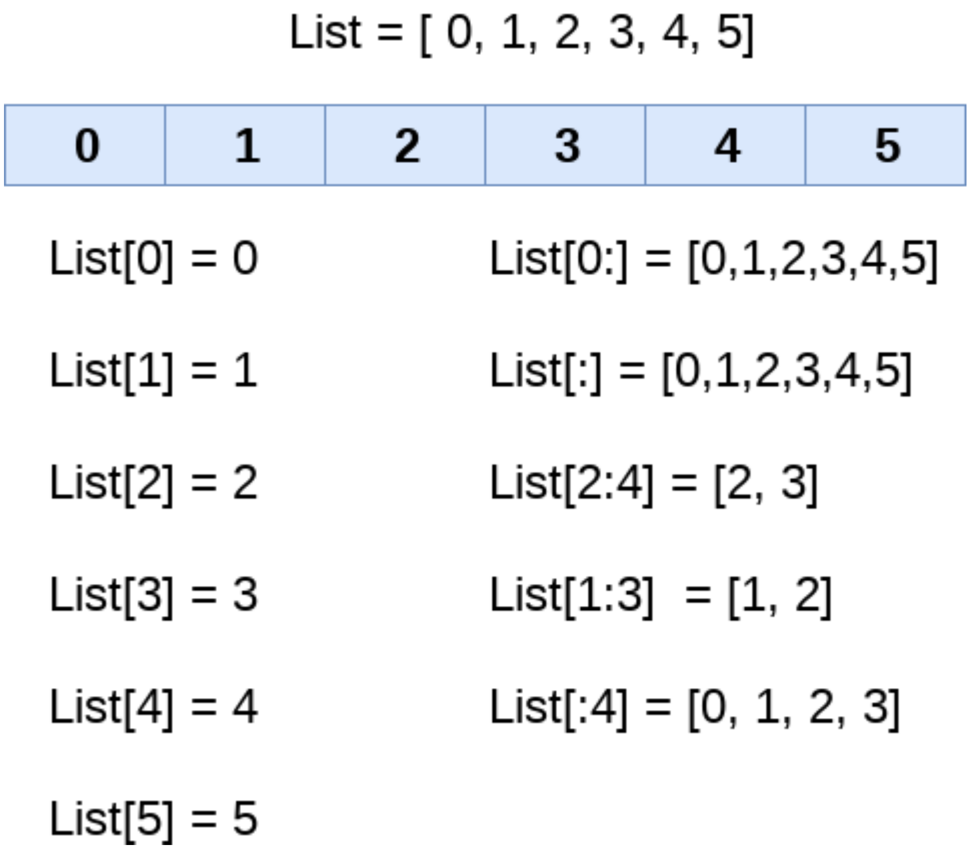
Output:

```
True
```

List indexing and slicing:

The indexing is processed in the same way as it happens with the strings. The elements of the list can be accessed by using the slice operator [].

The index starts from 0 and goes to length - 1. The first element of the list is stored at the 0th index, the second element of the list is stored at the 1st index, and so on.



We can get the sub-list of the list using the following syntax.

list_varible(start:stop:step)

- The **start** denotes the starting index position of the list.
- The **stop** denotes the last index position of the list.
- The **step** is used to skip the nth element within a **start:stop**

Consider the following example:

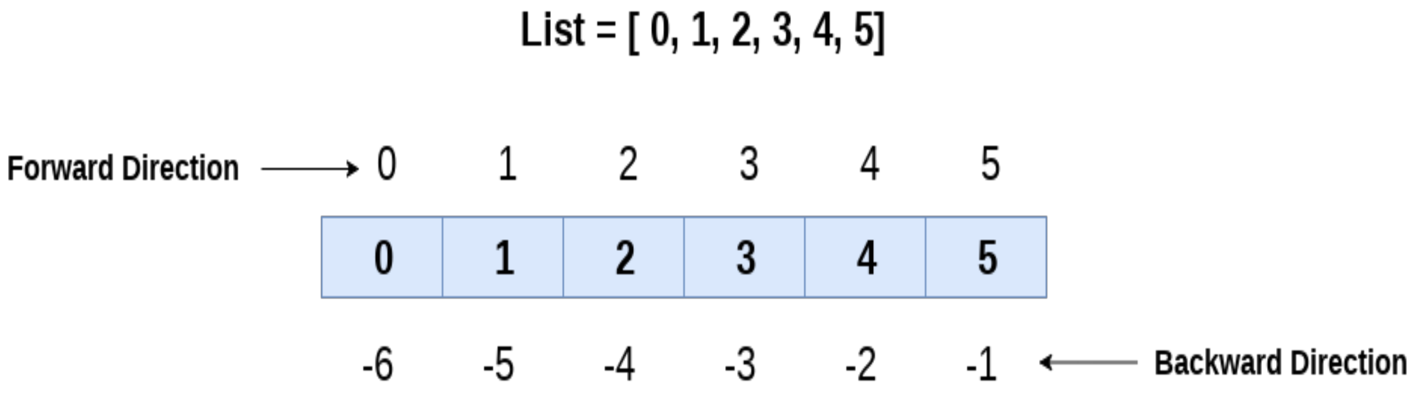
```
list = [1,2,3,4,5,6,7]
print(list[0])
print(list[1])
print(list[2])
print(list[3])
```

```
# Slicing the elements
print(list[0:6])
# By default the index value is 0 so its starts from the 0th element and go for index -1.
print(list[:])
print(list[2:5])
print(list[1:6:2])
```

Output:

```
1
2
3
4
[1, 2, 3, 4, 5, 6]
[1, 2, 3, 4, 5, 6, 7]
[3, 4, 5]
[2, 4, 6]
```

Unlike other languages, Python provides the flexibility to use the negative indexing also. The negative indices are counted from the right. The last element (rightmost) of the list has the index -1; its adjacent left element is present at the index -2 and so on until the left-most elements are encountered.



Let's have a look at the following example where we will use negative indexing to access the elements of the list.

```
list = [1,2,3,4,5]
print(list[-1])
print(list[-3:])
print(list[:-1])
print(list[-3:-1])
```

Output:

```
5
[3, 4, 5]
[1, 2, 3, 4]
[3, 4]
```

As we discussed above, we can get an element by using negative indexing. In the above code, the first print statement returned the rightmost element of the list. The second print statement returned the sub-list, and so on.

Updating List values:
Lists are the most versatile data structures in Python since they are mutable, and their values can be updated by using the **slice** and **assignment** operator.
Python also provides **append()** and **insert()** methods, which can be used to add values to the list.

Consider the following example to update the values inside the list.

```
list = [1, 2, 3, 4, 5, 6]
print(list)
# It will assign value to the value to the second index
list[2] = 10
print(list)
# Adding multiple-element
list[1:3] = [89, 78]
print(list)
# It will add value at the end of the list
list[-1] = 25
print(list)
# It will append value at the end of the list
list.append(10)
print(list)
# It will insert value at index 2 in the list
list.insert(2,20)
print(list)
```

Output:

```
[1, 2, 3, 4, 5, 6]
[1, 2, 10, 4, 5, 6]
[1, 89, 78, 4, 5, 6]
[1, 89, 78, 4, 5, 25]
[1, 89, 78, 4, 5, 25,10]
[1, 89, 20,78, 4, 5, 25,10]
```

The list elements can also be deleted by using the **del** keyword. Python also provides us the **remove()** method if we do not know which element is to be deleted from the list.

Consider the following example to delete the list elements.

```
list = [1, 2, 3, 4, 5, 6]
print(list)
```

```
del list[1:3]
print(list)
# It will delete the values from index 1 to index 2
list.remove(6)
print(list)
# It will delete the first occurrence of value 6 from the list and it will throws an error if element is not present in the list
```

Output:

```
[1, 2, 3, 4, 5, 6]
[1, 4, 5, 6]
[1, 4, 5]
```

Python List Operations:

The concatenation (+) and repetition (*) operators work in the same way as they were working with the strings.

Let's see how the list responds to various operators.

Consider a Lists l1 = [1, 2, 3, 4], **and** l2 = [5, 6, 7, 8] to perform operation.

Operator	**Description**	**Example**
Repetition	The repetition operator enables the list elements to be repeated multiple times.	l1*2 = [1, 2, 3, 4, 1, 2, 3, 4]
Concatenation	It concatenates the list mentioned on either side of the operator.	l1+l2 = [1, 2, 3, 4, 5, 6, 7, 8]
Membership	It returns true if a particular item exists in a particular list otherwise false.	print(2 in l1) prints True.
Iteration	The for loop is used to iterate over the list elements.	for i in l1: print(i) **Output** 1 2 3 4
Length	It is used to get the length of the list	len(l1) = 4

Iterating a List:
A list can be iterated by using a **for - in** loop. A simple list containing four strings, which can be iterated as follows.

```
list = ["John", "David", "James", "Jonathan"]
for i in list:
# The i variable will iterate over the elements of the List and contains each element in each i
teration.
   print(i)
```

Output:

```
John
David
James
Jonathan
```

Adding elements to the list:
Python provides **append()** and **insert()** functions which are used to add an element to the list. Insert() function will insert the element to a particular index, however, the append() function can only add value to the end of the list.

Consider the following example in which, we are taking the elements of the list from the user and printing the list on the console.

```
#Declaring the empty list
l =[]
#Number of elements will be entered by the user
n = int(input("Enter the number of elements in the list:"))
# for loop to take the input
for i in range(0,n):
   # The input is taken from the user and added to the list as the item
   l.append(input("Enter the item:"))
print("printing the list items..")
# traversal loop to print the list items
for i in l:
   print(i, end = "  ")

l.insert(2,50)
print("printing the list items after insertion..")
for i in l:
   print(i, end = "  ")
```

Output:

```
Enter the number of elements in the list:5
Enter the item:25
Enter the item:46
Enter the item:12
Enter the item:75
Enter the item:42
printing the list items..
25  46  12  75  42
printing the list items after insertion..
25  46  50  12  75  42
```

Python List Built-in functions:

Python provides the following built-in functions, which can be used with the lists.

SN	Function	Description	Example
1	cmp(list1, list2)	It compares the elements of both the lists.	This method is not used in the Python 3 and the above versions.

2	len(list)	It is used to calculate the length of the list.	L1 = [1,2,3,4,5,6,7,8]
			print(len(L1))
			8

3	max(list)	It returns the maximum element of the list.	L1 = [12,34,26,48,72]
			print(max(L1))
			72

4	min(list)	It returns the minimum element of the list.	L1 = [12,34,26,48,72]
			print(min(L1))
			12

5	list.sort()	It sorts the list ascending by default.	cars = ['Ford', 'BMW', 'Volvo']
			print(cars.sort())
			['Ford', 'BMW', 'Volvo']

5	list(seq)	It converts any sequence to the list.	str = "Johnson"
			s = list(str)
			print(type(s))
			<class list>

Let's have a look at the few list examples:

Example: 1- Write a program to find the sum of the element in the list.

```
list1 = [3,4,5,9,10,12,24]
sum = 0
for i in list1:
    sum = sum+i
print("The sum is:",sum)
```

Output:

```
The sum is: 67
```

Example: 2- Write the program to find the common elements from lists.

```
list1 = [1,2,3,4,5,6]
list2 = [7,8,9,2,10]
for x in list1:
    for y in list2:
        if x == y:
            print("The common element is:",x)
```

Output:

```
The common element is: 2
```

Example: 3- Write the program to remove the duplicate elements of the list.

```
list1 = [1,2,2,3,55,98,65,65,13,29]
# Declare an empty list that will store unique values
list2 = []
for i in list1:
    if i not in list2:
        list2.append(i)
print(list2)
```

Output:

```
[1, 2, 3, 55, 98, 65, 13, 29]
```